OCR Computer Science A Level

1.2.1 Systems Software

Advanced Notes

Specification:

1.2.1 a)

- **Operating Systems**
  - The need for, function and purpose of operating systems

1.2.1 b)

- **Memory Management**
  - Paging
  - Segmentation
  - Virtual Memory

1.2.1 c)

- **Interrupts**
  - Role of interrupts
  - Role of ISR (Interrupt Service Routines)
  - Role of interrupts within Fetch-Decode- Execute Cycle

1.2.1 d)

- **Scheduling**
  - Round robin
  - First come first served
  - Multi-level feedback queues
  - Shortest job first
  - Shortest remaining time

1.2.1 e)

- **Types of operating system**
  - Distributed
  - Embedded
  - Multi-tasking
  - Multi-user
  - Real Time

1.2.1 f)

- **BIOS**

1.2.1 g)

- **Device drivers**

1.2.1 h)

- **Virtual machines**
  - Where software is used to take on the function of a machine for:
    - Executing intermediate code
    - Running an operating system within another

Operating Systems

The term ‘operating system’ refers to a collection of programs that work together to provide an interface between the user and computer. Operating systems enable the user to communicate with the computer and perform certain low-level tasks involving the management of computer memory and resources. Therefore they are essential in devices such as laptops, mobile phones and games consoles. Examples of popular desktop operating systems include Windows and macOS while popular mobile phone operating systems include iOS and Android.

Operating systems are essential to a computer system as they provide the following features:

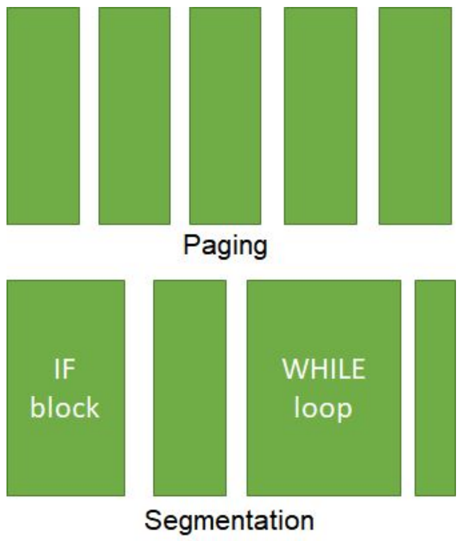
- Memory management (*paging, segmentation, virtual memory*)
- Resource management (*scheduling*)
- File management (*moving, editing, deleting files and folders*)
- Input/ Output management (*device drivers*)
- Interrupt management
- Utility software (*disk defragmenter, backup, formatting etc.*)
- Security (*firewall*)
- User interface

Memory Management

Computer memory must be shared fairly between multiple programs and applications being used. Often, main memory is not large enough to store all of the programs being used. One of the roles of the operating system is to ensure the main memory is shared efficiently between programs through the use of paging, segmentation and virtual memory.

Paging

Paging is when memory is split up into equal-sized sections known as pages, with programs being made up of a certain number of equally-sized pages. These can then be swapped between main memory and the hard disk as needed.



Segmentation

Segmentation is the splitting up of memory into logical sized divisions, known as segments, which vary in size. These are representative of the structure and logical flow of the program, with segments being allocated to blocks of code such as conditional statements or loops.

Virtual Memory

Virtual memory uses a section of the hard drive to act as RAM when the space in main memory is insufficient to store programs being used. Sections of programs that are not currently in use are temporarily moved into virtual memory through paging, freeing up memory for other programs in RAM.

The key issue with using these three techniques is disk thrashing. This is when the computer 'freezes' and occurs as a result of pages being swapped too frequently between the hard disk and main memory. As a result, more time is spent transferring these pages between main memory and the hard disk then is spent actually running the program. This issue becomes progressively worse as virtual memory is filled up.

Interrupts

Interrupts are signals generated by software or hardware to indicate to the processor that a process needs attention. Different types of interrupts have different priorities and how urgent they are must be taken into account by the operating system when allocating processor time. Interrupts are stored in order of their priority within an abstract data structure called a priority queue in a special register known as an interrupt register. It is the job of the operating system to ensure interrupts are serviced fairly by the processor through the Interrupt Service Routine. Examples of interrupts include a printer signalling the completion of a print job or a peripheral signalling power failure.

Interrupt Service Routine

The processor checks the contents of the interrupt register at the end of each Fetch-Decode-Execute cycle. If an interrupt exists that is of a higher priority to the process being executed, the current contents of the special purpose registers in the CPU are temporarily transferred into a stack. The processor then responds to the interrupt by loading the appropriate interrupt service routine (ISR) into RAM. A flag is set to signal the ISR has begun.

Synoptic Link

Abstract **data structures** such as **queues** and **stacks** are discussed further in 1.4.2 Data Structures. A queue is a **FIFO (First In First Out)** data structure in which the first element in the list is the first to leave. A stack is a **LIFO (Last In First Out)** data structure in which the first element in the list is the last to leave.

Once the interrupt has been serviced, the flag is reset. The interrupt queue is checked again for further interrupts of a higher priority to the process that was originally being executed.

If there are more interrupts to be serviced, the process described above is repeated until all priority interrupts have been serviced. If there are no more interrupts or interrupts are of a lower priority to the current process, the contents of the stack are transferred back into the registers in memory. The Fetch-Decode-Execute cycle resumes as before.

Synoptic Link

The memory registers referred to here are those you have encountered in 1.1.1; the PC, MAR, MDR, CIR and ALU!

Scheduling

Another core function of the operating system is to ensure all sections of programs being run (known as 'jobs') receive a fair amount of processing time. This is done by implementing various scheduling algorithms, which work in different ways. Scheduling algorithms can either be:

1. Pre-emptive
   Jobs are actively made to start and stop by the operating system.
   For example: Multilevel Feedback Queues, Shortest Remaining Time, Round Robin
2. Non pre-emptive
   Once a job is started, it is left alone until it is completed.
   For example: First Come First Served, Shortest Job First

Note

The scheduling algorithm used depends on the task at hand. For example, **Shortest Remaining Time** is used where the completion of shorter tasks before other tasks is preferred. Typically however, operating systems such as Windows use **multilevel feedback queues.**

Below are the scheduling algorithms you need to know:

- Round robin
  Each job is given a section of processor time - known as a time slice - within which it is allowed to execute. Once each job in the queue has used its first time slice, the operating system again grants each job an equal slice of processor time. This continues until a job has been completed, at which point it is removed from the queue. Although Round Robin ensures each job is seen to, longer jobs will take a much longer time for completion due to their execution being inefficiently split up into multiple cycles. This algorithm also does not take into account job priority.

- First come first served

  Jobs are processed in chronological order by which they entered the queue. Although this is straightforward to implement, FCFS again does not allocate processor time based on priority.
- Multilevel feedback queues

  This makes use of multiple queues, each which is ordered based on a different priority.
  This can be difficult to implement due to deciding which job to prioritise based on a combination of priorities.
- Shortest job first

  The queue storing jobs to be processed is ordered according to the time required for completion, with the longest jobs being serviced at the end.
  This type of scheduling is most suited to batch systems, where shorter jobs are given preference to minimise waiting time.
  However it requires the processor to know or calculate how long each job will take and this is not always possible. There is also a risk of processor starvation if short jobs continue being added to the job queue.
- Shortest remaining time

  The queue storing jobs to be processed is ordered according to the time left for completion, with the jobs with the least time to completion being serviced first.
  Again, there is a risk of processor starvation for longer jobs if short jobs are added to the job queue.

Processor starvation

When a particular process does not receive enough processor time in order to execute and be completed.

Types of Operating System

There are various types of operating systems which have different uses.

- Distributed

  This is a type of operating system which is run across multiple devices, allowing the load to be spread across multiple computer processors when a task is run.
- Embedded

  Built to perform a small range of specific tasks, this operating system is catered towards a specific device. They are limited in their functionality and hard to update although they consume significantly less power than other types of OS.

Note

Embedded systems are common in consumer appliances such as coffee and washing machines as well as in-car systems.

- Multi-tasking

  Multi-tasking operating systems enable the user to carry out tasks seemingly simultaneously. This is done by using time slicing to switch quickly between programs and applications in memory.
- Multi-user

  Multiple users make use of one computer, typically a supercomputer, within a multi-user system. Therefore a scheduling algorithm must be used to ensure processor time is shared fairly between jobs. Without a suitable scheduling algorithm, there is a risk of processor starvation, which is when a process is not given adequate processor time to execute and complete.
- Real Time

  Commonly used in time-critical computer systems, a real time OS is designed to perform a task within a guaranteed time frame. Examples of use include the management of control rods at a nuclear power station or within self-driving cars: any situation where a response within a certain time period is crucial to safety.

BIOS

The Basic Input Output System is the first program that runs when a computer system is switched on. The Program Counter register points to the location of the BIOS upon each start-up of the computer as the BIOS is responsible for running various key tests before the operating system is loaded into memory, such as:

- POST (Power-on self test) which ensures that all hardware (keyboards, disk drives) are correctly connected and functional
- Checking the CPU clock, memory and processor is operational
- Testing for external memory devices connected to the computer

The BIOS is critical to the computer system as it is only after these checks are completed that the operating system can be loaded into RAM from the hard disk.

Bootstrap/ bootloader

... is the name given to the program that loads the operating system from the hard disk into main memory.

Device Drivers

Device drivers are computer programs which are provided by the operating system and allow the operating system to interact with hardware.

When a piece of hardware is used, such as a keyboard, it is the device driver that communicates this request to the operating system which can then produce the relevant output - which in this case is displaying a letter on the screen.

Device drivers are specific to the computer's architecture, so different drivers must be used for different device types such as smartphones, games consoles and desktop PCs. As drivers interact with the operating system, they are also specific to the operating system installed on the device.

Synoptic Link

Device drivers are an example of abstraction as you as a user are able to interact with hardware without knowing details about how this works. This concept is explored further in 2.2.2

Virtual Machines

A virtual machine is a theoretical computer in that it is a software implementation of a computer system. It provides an environment with a translator for intermediate code to run.

Synoptic Link

Translators are discussed further in 1.2.2.

Intermediate Code

Code that is halfway between machine code and object code is called intermediate code. This is independent of the processor architecture so can be used across different machines and operating systems.

Virtual machines are commonly used to create a development environment for programmers to test programs on different operating systems. The advantage of this is that it saves both the time and money of having to purchase multiple devices solely for testing. However, running intermediate code in a virtual machine can also be considerably slower compared to running low-level code on the device it was designed for.

Other uses of virtual machines include:

- Protection from malware

  Malware will affect the virtual machine rather than the device being used.

- Running incompatible software

  Programs specific to different operating systems or different versions of an operating system can be run within a VM, saving time and money required to purchase the hardware.

  A common example is of games consoles being implemented on PCs via a virtual machine.